Originally published in issue 1 of (IN)SECURE Magazine, get it for free in PDF format at www.insecuremag.com

An Introduction to Securing Linux with Apache, ProFTPd, and Samba

by Zach Riggle

While the vast majority of Linux users are hard-core techies, some may be using Linux because they want to try something new, are interested in the technology, or simply cannot afford or do not want to use Microsoft Windows. After becoming acquainted with the new interface of Linux, whether KDE, Gnome, or another window manager, users may begin to explore their system. Many machines come with default installations of Apache and Samba, and a few others even include a FTP daemon.

While these services may be disabled by default, some users may be inclined to use these programs. This article is a brief, but in-depth tutorial on how to keep these applications up-to-date and secure.

1. Installation

First off, we'll help out those that do not have the proper software installed. Your particular distribution of Linux may include a software management system that allows for easy installation of new software (i.e. YaST for SuSE, RPM for RedHat, and Slackpkg for Slackware). Many others feel more comfortable compiling software. *Compiling* is the feeding of programming code to a *compiler* program, which in turn creates the actual programs, or *binaries*, that the user actually runs. If you feel more comfortable using your Linux distribution's custom software management, then feel free to get the software that way. If it is either not offered, or you wish to try your hand at or already know how to compile, read on.

Note: When downloading software source code, it is wise to store it all in one place, so that you can find it again easily. If you already have the source on your computer, you can use a 'patch' to update the source to the latest version, without downloading the whole thing. The standard directory for this is /usr/src, but you may use any directory you wish.

Since this is a security article, it is very important that I go over the process of verifying files that you download. When you are downloading files, you will commonly see a "GPG Signature", `.asc`, or "md5 sum" nearby. These are all methods of verifying that the file you are downloading is actually the file that you think you are downloading. Every once in a while, somebody will try to make a few changes to something before they

put it on a mirror, or someone will compromise the server and put some bad things in what you are downloading. These files, or MD5 sums, allow you to be absolutely positive that you are downloading a file that still has its integrity (meaning it has not been modified).

The easiest way of making sure the file is what it is supposed to be is the `md5sum` command. Just type

```
md5sum <the-file-in-question>
```

And the MD5 sum of that file will be displayed on your console. If that matches the displayed MD5 sum, then all systems are "Go." Another common method is GnuPG\PGP signatures. GnuPG stands for Gnu Privacy Guard, and is the open-source equivalent of PGP (Pretty Good Privacy). If you do not already have GnuPG, you can get it from http://www.gnupg.org. There are two parts involved in verifying files with GnuPG — importing the GnuPG key, and verifying the file. These should be freely available on whatever website you are downloading from, and are probably on the same page. If you cannot find a link for the file signature, try downloading "<the-file-in-question>.asc". After downloading the file in question, enter the following:

```
gpg --import their-key-file.asc
gpg --verify their-verification-file.asc
```

These commands should by synonymous with PGP. The verification file will often have the same file name as the file you downloaded, with ".asc" on the end of the file name. If you get an error stating something about "no signed data" or "file open error", then the file that you downloaded was a gzip'd or bzip'd tar archive; the `.asc` file was made for only the .tar archive. `gunzip` (or `bunzip`) the tar archive, and then try again.

1A. Apache

The first software package that we're going to install is Apache. Apache is, quite literally, the world's most popular web server. Due to its popularity and widespread support, it is very well-documented, easy-to-use, and secure. To download Apache, go to http://httpd.apache.org and select one of the mirrors. You want to download Apache 1.3.

Now, open up a terminal, and navigate to the directory that you saved the file in. To extract the file, we're going to use the command:

```
$ tar -xvzf apache_1.3.XX.tar.gz
```

Where XX is the sub-version of Apache (as of this article, the latest version is 1.3.33). This command will extract Apache into a sub-directory of the same name as the archive. Change to that directory:

```
$ cd apache_1.3.XX
```

Now, we are ready to start compiling the software. First, we must tell it to configure itself for our system.For the purpose of this article, we are going to install Apache to the default `/usr` directory. If you would rather install somewhere else, feel free, but replace `/usr` with your chosen directory.
To start configuration:

```
$ ./configure \
  --prefix=/usr \
  --enable-static \
  --enable-shared \
  --sysconfdir=/etc/apache
```

This command tells the software that you want it installed to the `/usr` directory, and want it to be able to use shared and static libraries. Its configuration files will be located in `/etc/apache`.

Note: I use a '\' after each argument because it makes it easier to read. You do not have to do this — everything can be written on one line. Just remove the '\'es and put everything on one line, and it will work the exact same. For example, the equivalent to the above command is:

```
./configure --prefix=/usr --enable-static --enable-shared --sysconfdir=/etc/apache
```

As you can see, it's a bit hard to fit everything on one line, on paper. I had to reduce the font size so that everything would fit. In your console, it will just wrap around, but it's hard to tell where the text is wrapping around, and where you actually need to press enter, if it is on paper.

After the configuration has been completed, type the following command into your terminal:

```
$ make
```

This process may take a while, depending on the speed of your computer. After it has completed, 'su' as root:

```
$ su root
```

Install and start Apache:

```
# make install
# apachectl start
```

Congratulations, you have just installed Apache. However, it is only set up with its default configuration. We will set up the rest of the software before we begin configuration.

1B. Samba

The installation of Samba is very similar to the installation of Apache. Open your browser to http://www.samba.org/, and download the latest version of Samba.

Extract (after verifying it with GPG) the `samba-latest.tar.gz` archive the same way you did with the commands. Then change into that directory, and type the following commands into your console:

```
$ ./configure --enable-cups \
  --enable-static=no \
  --enable-shared=yes \
  --with-fhs \
  --prefix=/usr \
  --localstatedir=/var \
  --bindir=/usr/bin \
  --sbindir=/usr/sbin \
  --sysconfdir=/etc \
  --with-configdir=/etc/samba \
  --with-privatedir=/etc/samba/private
$ make
$ su
# make install
```

This will set up your Samba installation for easy configuration. All of the configuration files will be installed to `/etc/samba`.

1C. ProFTPd Installation

Once again, the installation is very simple. Go to http://www.proftpd.org, and download the latest version of ProFTPd by clicking the 'gz' button in the upper-left corner.

Extract the .tar.gz file in the same manner as before, and type the following command into the terminal, once you have changed to the extracted directory:

```
$ ./configure \
  --prefix=/usr \
  --sysconfdir=/etc/proftpd/ \
  --enable-autoshadow
$ make
$ su
# make install
```

And you're done! There are many articles on configuring Apache, Samba, and ProFTPd, so this article will not cover that particular aspect. This section was only designed to get you up-and-running with the current software.

2. Securing Apache

Apache is probably the piece of software you should be the most worried about being exploited. It is not insecure, but as it is used worldwide, and the servers are usually accessible to the public, there are more bad-guys looking for holes in the software. For the purpose of this article, we will assume that your configuration files are located in `/etc/apache`.

The first thing that one should do with any Apache installation, is make sure that it is not running as `root`. Running as root can lead to many security exploits turning out to be their worst. We can remedy this by opening `httpd.conf` and finding the `User` directive.

Open `httpd.conf`, and find the User directive. On a default installation, it might read

```
User nobody
Group nobody
```

We are going to change this. Close `httpd.conf`, and type the following command into the console:

```
Groupadd apacheuser
Useradd -g httpd -c Apache User -p <some password> apacheuser
```

This will create a user `apacheuser` that belongs to the group apacheuser. It may also be wise to change the permissions so that the user `apacheuser` is in control of all of your web-server's documents:

```
chown -R apacheuser.apacheuser \
     /path/to/your/htdocs/and/cgi-bin
```

So that `apacheuser` can access your web-server's documents. Now, open `httpd.conf` again, and change the User and Group directive to:

```
User apacheuser
Group apacheuser
```

You may also notice that, if you scroll down, there are many other sections of the `httpd.conf` file. The one that we are going to check out next is the `<Directory>` directive.

On the default installation of Apache, you can get a list of files in a given directory if it does not contain an Index file (default Index files are index.htm and index.html). This could potentially lead to people seeing files that they are not supposed to see. In order to prevent this, open `httpd.conf` and search for the following string:

```
<Directory />
```

Directly *before* that line, insert the following directives:

```
Options -Indexes
```

This will prevent users from getting a nice, prettified list of files in a directory that they can sort, if they want to. Since it is not *within* the <Directory> statement, it applies to all directories, unless specified otherwise. To turn it back on for a specific directory, just place this line inside of a <Directory> statement.

If the directory you wish to protect is not your root web-document directory (for example, http://blah.com/ would be the root directory. http://blah.com/subdirectory/ is not), you have to create a new `<Directory>` directive. To do this, open up `httpd.conf`, and find the section that looks something like this:

```
<Directory />
…Some configuration information…
</Directory>
```

That denotes the configuration for your root document directory. Directly after that, create a similar section, but replace / with the path to your directory, like so:

```
<Directory /path/to/my/directory>
</Directory>
```

This tells Apache that there are some special rules for your directory. To set up the basics, enter the following into

3. Securing ProFTPd

ProFTPd is a little bit easier to configure, since is it much simpler software. All that FTP software does is allow the transfer of files between two machines. There are many other ways of doing this, such as SCP and secure Samba (which I will go over later in this article).

First, let's open up `/etc/proftpd/proftpd.conf`. The first line of configuration reads:

```
ServerName "ProFTPD Default Installation"
```

It is very easy to tell why we don't want this there. Not only does it give away what software we are using to any potential attackers, it also lets them know that this box may not be particularly secure — simply because we did not remove the "Default Installation" text from that line. Change that to something else, or try to confound any attackers by replacing it with the name of another FTP Daemon. Common FTP daemons include Wu-FTPd, NcFTPd, and vsFTPd.

The next thing we may wish to do is change the umask. A 'umask' is simply the inverse of the default mask we wish to assign to a file that is uploaded. The default is `022`, so all files will be `chmod`'ed `755`. Since this declares the uploaded file executable, this is bad. Change `022` to `133`. Now files are readable, but not executable.

Further down, you may notice a `MaxInstances` statement. This simply determines the maximum number of FTPd instances running on any machine at a given time. The default is 30, and this is plenty for any server. For a smaller, personal server, you may wish to set this to 2-5 (1 is not suggested, because if you accidentally get disconnected, you will have to wait a while to reconnect). On slower machines, this is especially important, because it prevents attackers from connecting to your box multiple times, and causing ProFTPd to spawn 30 processes — which may not be all too healthy for the box.

Similar to the Apache setup, it is highly advised that you set up an account for ProFTPd to run as. The default is `nobody` of `nogroup`. You could potentially use the same user as the Apache installation, but this is not suggested, because if ProFTPd is exploited, the attacker could also mess with your Web documents and CGI scripts. Exit `proftpd.conf`.

Now open `/etc/ftpusers` in an editor. This file contains a list of all of the users that *cannot* log into your FTP server. Those that should always be in this list include `root` and any other super-users on your system. Since Apache won't be logging into your FTP server (unless someone exploited Apache), place that username in the file too. If you do not wish to have anonymous logins (a definite security breach potential), enter the user `ftp` into the list.

4. Securing Samba

In order to have complete security, *everything* on your box must be locked down. The easiest way to lock down a service is to limit who can access it. This is exactly what we are going to do with Samba. Open up `/etc/samba/smb.conf`. Look for the directive `hosts allow`. Change this from its default to:

```
hosts allow 127.0.0.1
```

This will ensure that only localhost can connect to any Samba shares. At first, one would think that this kind of defeats the purpose of Samba. This is not true, when combined with SSH tunneling. Tunnels are extremely easily set up in Putty, but since most users of this article will be running Linux, I will show how to do it from the command-line. It should be noted that with this method, you *must* keep the SSH session open for as long as you want to connect to the samba share.

Note: Think of a tunnel as encrypted port forwarding, that bypasses the router. For this to function properly, SSH has to be accessible. You must set up your router or firewall to allow SSH traffic to the Samba-hosting server.

```
$ ssh -L 139:127.0.0.1:10139 \
  -l [Login Name] \
  -N \
  [Samba Server's IP]
```

This will create a secure "tunnel" from your computer to the Samba server, and not execute any commands after logging in. You do not get a shell — only port forwarding is set up. In this situation, if you connect to port 10139 on your *local* machine, all data sent to\from it will also be sent to the Samba machine, which will make a connection back to itself on port 139, and forward all data back and forth between the machines. Now, try to access the Samba share at 127.0.0.1:10139. Note that on Windows machines, this does not work. Windows will only let you access Samba shares via port 139, for security measures. This can be circumvented by installing a Loopback adapter with its own private IP address. That is not within the scope of this article, and the topic can easily be found by Googling for it.

Samba also has a large potential for setting up user-based authentication, and only allowing certain users to connect to certain shares. This is quite trivial to set up, so instructions will not be included here.

5. Chroot Environments

Linux has one large advantage that Microsoft Windows servers do not — chroot environments. For those that are not familiar with the 'chroot' command, it essentially changes the root directory for whatever process is using it. Those that are already familiar with the command should skip section 5A.

5A. Explanation of Chroot

A chroot is simply a stripped-down version of your box. It has access to an extremely limited number of files and utilities, and you can't really do much with it. This is the idea behind a chroot — you can limit whatever is inside it to do only exactly what it needs to do, and nothing else. So ProFTPd, even if it is compromised, in a chroot, cannot possibly give out your `shadow` unless you specifically place it in the chroot.

To illustrate the idea of a chroot, create a script with the following in it:

```
#!/usr/bin/bash
CHROOT=/tmp/chroot

mkdir $CHROOT
mkdir $CHROOT/bin
mkdir $CHROOT/lib
cp -a /bin/bash $CHROOT/bin
cp -a /bin/ls   $CHROOT/bin
cp -a /lib/ld-*.so*            $CHROOT/lib
cp -a /lib/librt*.so*          $CHROOT/lib
cp -a /lib/libc.so*            $CHROOT/lib
cp -a /lib/libc-*.so*          $CHROOT/lib
cp -a /lib/libdl*.so*          $CHROOT/lib
cp -a /lib/libtermcap*.so*     $CHROOT/lib
cp -a /lib/libpthread*.so*     $CHROOT/lib
chroot $CHROOT bash
```

Now, `chmod +x` the script, and run it as `root`. You will probably be given a prompt that looks like this:

```
bash-3.00#
```

If you browse around the file system, you will find that it only consists of the files that we just placed in `/tmp/chroot`, via the script. In fact, you cannot access any files that *aren't* in `/tmp/chroot`. This is because you are in a 'chroot jail'. Try as you like, you cannot leave the chroot jail without killing `bash` via `exit` or killing the process. Either way, `bash` dies. Assuming that instead of `bash`, the program in the chroot jail was Apache, an attacker should not be able to escape the chroot jail without killing whatever process it is they are using to access your system — which they can't do without losing access to your system. So even if your box *is* compromised, the attacker is stuck in a very small, very limited chroot environment.

5B. Installing Apache into a chroot

This is section is designed to be an example of how to install software into a chroot. It usually follows a pattern like so:

1. Copy configuration files
2. Copy binaries

3. Set permissions

That's about all there is to it. Many people get lost among all the file-copying, and decide it is not something they wish to use. Trust me, it is well worth your while to install and properly configure a chroot. Once you understand what it is you are trying to do, everything makes sense. I only give exact instructions for installing Apache, but installing ProFTPd into a chroot would not be hard. (It would also be something that may or may not be useful, depending on your uses for FTP). Remember — something inside a chroot cannot access files outside of it. If you install ProFTPd into a chroot, and the user's home directories are *not* in the chroot, there may be problems).

As I stated before, there is a pattern to setting up a chroot. For those of you familiar with a "Fuzzy clock" (a "Fuzzy clock" is a clock that does not give the *exact* time. For example, if the time is 8:23PM, a Fuzzy clock might say "Late"), let this be the "Fuzzy procedure":

1. Copying files.
2. Copying more files.
3. Setting permissions

1. 1. Copying files.

The most confusing part about configuring a chroot environment is knowing which files should be copied into the chroot, and why. Since two Linux systems are almost never the same, the article makes the following assumptions (some have been stated before, I am just reiterating):

- Your web documents are in `/var/www/htdocs` and `.../cgi-bin`
- Your Apache configuration files are in `/etc/apache`.
- Apache was installed with `--prefix=/usr`.
- Your chroot environment will be located in `/chroot/`.
- The user account under which Apache runs is named apacheuser.
- You are using Apache version 1.3.
- Mod SSL has not been installed.
- Your system init scripts are located in `/etc/rc.d`. (This is the default for many distributions, but some may boxes place them in `/etc/rc.d/init.d`)

Let's start the setting up of the chroot! Just so that it catches your eye (and you don't do anything by mistake) directories in your chroot will be **blue**, while directories on your normal file system will be **red**. Accidentally deleting or modifying files could seriously mess up your system! I suggest you back up everything that is critical to your system (especially `/etc/`). You have been warned!

Setting up directories

```
# CHROOT=/chroot
# APACHE=$CHROOT/apache
# export CHROOT
# export APACHE

# mkdir $CHROOT
# mkdir $APACHE
# mkdir $APACHE/dev/
# mkdir $APACHE/etc/
# mkdir $APACHE/etc/apache
# mkdir $APACHE/home
# mkdir $APACHE/home/apacheuser
# mkdir $APACHE/lib/
# mkdir $APACHE/usr
# mkdir $APACHE/usr/bin
# mkdir $APACHE/usr/sbin
# mkdir $APACHE/usr/lib
# mkdir $APACHE/usr/libexec
# mkdir $APACHE/var/
# mkdir $APACHE/var/log/
# mkdir $APACHE/var/log/apache
# mkdir $APACHE/var/www/
```

Setting up devices. This creates a `null` device:

```
# mknod $APACHE/dev/null  c 1 3
# chown root.root $APACHE/dev/null
# chmod 666 $APACHE/dev/null
```

The `null` device is simply a device that has nothing in it. It does not contain any information, even if you write it there. This is what's so wonderful about it. Let's say there's an error message that we don't want to see — just redirect it to a file and you won't see it anymore. However, the data is actually written to that file. However, if you redirect it to `/dev/null`, the data is still redirected, but it does not get written to file. Also, if you `cat /dev/null` into a file, you will get a file with a length of zero, or erase any data in the file.

Setting up `/etc/`. You will want to modify `passwd`, `shadow`, and `group` so that only `apacheuser` is in the first two, and only groups that `apacheuser` is part of remains in group.

```
# cp -a /etc/passwd $APACHE/etc/
# cp -a /etc/group $APACHE/etc/
# cp -a /etc/shadow $APACHE/etc/
```

Now, we will begin copying Apache and its related files. Run `apachectl stop` to turn off Apache.

```
# cp -Ra /etc/apache $APACHE/etc/
# cp -Ra /var/www/ $APACHE/var/
# cp -a  /usr/sbin/httpd $APACHE/usr/sbin/
# cp -a /usr/sbin/apachectl $APACHE/usr/sbin
# cp -Ra /usr/libexec/apache $APACHE/usr/libexec
```

Copy network-related files to the chroot.

```
# cp -a /etc/hosts $APACHE/etc/
# cp -a /etc/host.conf $APACHE/etc/
# cp -a /etc/resolv.conf $APACHE/etc/
# cp -a /etc/nsswitch.conf $APACHE/etc/
```

Since several files in `/etc/` won't change often (if at all) unless an attacker changes them, set the immutable bit on the files. This means that the files cannot be modified unless root removes the immutable bits from them first.

```
# chattr +i $APACHE/etc/passwd
# chattr +i $APACHE/etc/shadow
# chattr +i $APACHE/etc/group
# chattr +i $APACHE/etc/hosts
# chattr +i $APACHE/etc/host.conf
# chattr +i $APACHE/etc/resolv.conf
# chattr +i $APACHE/etc/nsswitch.conf
```

Also, in order for files to be written with the correct timestamp and zone information, you must copy that zone information into your chroot.

```
# cp /etc/localtime $APACHE/etc/
```

We must also tell syslog to monitor the Apache files. Syslog only monitors /var/log by default, and Apache's log files are in /chroot/apache/var/log. We must tell syslog to watch these, too. We can do this by opening up /etc/rc.d/rc.syslog and change "daemon syslogd -m 0" to "daemon syslogd -m 0 –a /chroot/apache/var/log". Also, we must change

```
echo -n "/usr/sbin/syslogd"
/usr/sbin/syslogd
```

to:

```
echo -n "/usr/sbin/syslogd"
/usr/sbin/syslogd -m 0 -a /chroot/apache/var/log
```

2. Copying More Files

Now comes the least exciting part: copying libraries. For determining what libraries a given executable file uses, we are going to use the utility `ldd`. `ldd` should come with the

GNU C\C++ Libraries. If it does not already have `ldd` or the GNU C\C++ Libraries, you can get them from www.gnu.org.

Now that you have `ldd` (or if you had it already), run the following command:

```
# ldd $APACHE/usr/sbin/httpd
```

The output of the command will look something like this:

```
libm.so.6 => /lib/libm.so.6 (0x4001f000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x40042000)
libdb-3.3.so => /lib/libdb-3.3.so (0x40071000)
libexpat.so.0 => /usr/lib/libexpat.so.0 (0x40105000)
libdl.so.2 => /lib/libdl.so.2 (0x40125000)
libc.so.6 => /lib/libc.so.6 (0x40129000)
/lib/ld-linux.so.2 (0x40000000)
```

This is a list of all of the libraries that Apache uses. You must copy the ones displayed by your ldd into $APACHE/lib/. Otherwise, Apache will not function. Since most of the libraries are referenced by symlinks, it is very highly suggested you do something like the following:

```
# cp /lib/libm* $APACHE/lib/
# cp /lib/libcrypt* $APACHE/lib/
# cp /lib/libdb* $APACHE/lib/
# cp /usr/lib/libexpat* $APACHE/lib/
# cp /lib/libdl* $APACHE/lib/
# cp /lib/libc* $APACHE/lib/
# cp /lib/ld-* $APACHE/lib/
```

Some libraries that you may not necessarily need (especially using the libc* wildcard) may be copied into the chroot. You can either remove them by hand, or leave them. A few extra libraries in your chroot will most likely not hurt anything, or provide any venues to a security breach.

3. Setting Permissions

This is probably the least fun part of configuring a chroot. Run the following script, or the equivalent of it:

```
#!/usr/bin/bash
touch $APACHE/var/log/apache/access_log
touch $APACHE/var/log/apache/error_log
chmod 600 $APACHE/var/log/apache/*
chattr +a $APACHE /var/log/apache/*
chmod 750 $APACHE/var/log/apache/
chmod 600 $APACHE/var/log/apache/*
chattr +a $APACHE/var/log/apache/*
```

```
chown -R root $APACHE
chmod -R 0755 $APACHE
```

Let 'er rip!

If you did everything correctly, Apache should now be ready to run from your chroot environment. Start it up and see if everything works out like it should!

```
chroot /chroot/apache/ /usr/sbin/httpd
```

Now, test your HTTP server and make sure everything is working correctly. If it is not, the following command will be useful in helping you track down the problem.

```
strace chroot /chroot/apache/ /usr/sbin/httpd 2> httpd.strace
```

This will put some debug information into a file named `httpd.strace`. Once everything is running correctly, you can remove your original Apache installation. If you are told something about `some-library.so`, then a module loaded by Apache is missing a library. Simply copy the library to the chroot, like shown above. Be sure to check both /lib and /usr/lib for the library.

6. Conclusion

Congratulations, you have just taken the first step in securing your Linux box, by securing its most often-attacked services. This is not and was not intended to be an all-inclusive security article. There are a lot of other aspects of security that should be considered. This article addressed some of the most common ones. The only way to do so is to seek the knowledge.